(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開平10－177495
(43)公開日　平成10年(1998)６月30日

(51)Int.Cl.[6]	識別記号	ＦＩ	
Ｇ０６Ｆ　9/46	３６０	Ｇ０６Ｆ　9/46	３６０Ｆ
			３６０Ｃ
13/00	３５７	13/00	３５７Ｚ

審査請求　未請求　請求項の数６　ＯＬ　（全 14 頁）

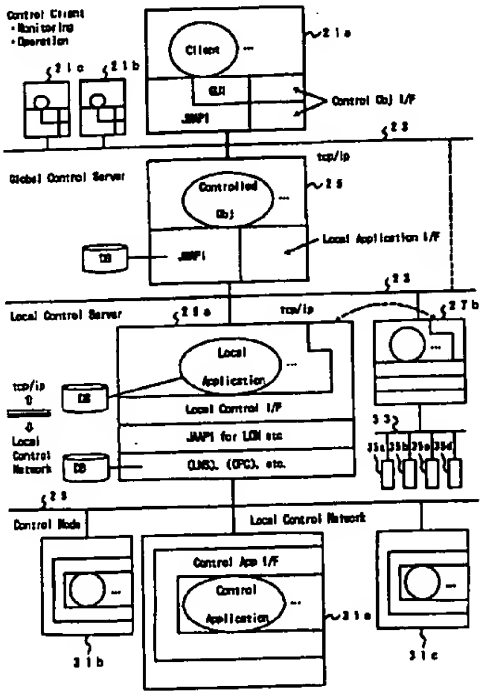
(21)出願番号　特願平8－338490

(22)出願日　平成８年(1996)12月18日

(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地
(72)発明者　竹内　陽一郎
東京都府中市東芝町１番地　株式会社東芝府中工場内
(74)代理人　弁理士　鈴江　武彦　（外６名）

(54)【発明の名称】　分散形制御ネットワークシステム

(57)【要約】
【課題】異なるローカルコントロールネットワークに接続されたコントロールノードを統一的に制御し、異なるネットワークに接続されたコントロールノード間での通信を可能にするとともに、データフォーマットを標準化し、１対複数のプログラム間通信機構を可能にすることを主要な課題とする。
【解決手段】分散形制御ネットワークシステムにおいて、ローカルコントロールサーバは、ユーザがＪａｖａで記述した各ノードに対する制御内容を実行時にそのノードで実行可能なコードに変換するジャスト・イン・タイム・コンパイラを有する。さらに、ローカルコントロールサーバは自系のネットワーク内のノードと、通信プロトコルの異なる他系のネットワーク内のノードとの通信を可能にするようにプロトコル変換機能を有する。また、各ノードが実行すべき制御内容及びノード間での処理の流れをＧＵＩ環境下でプログラミングする手段を提供する。



【特許請求の範囲】
【請求項１】　第１の通信プロトコルにより第１通信ネットワークに接続されたコントロールクライアントと：第２の通信プロトコルにより第２通信ネットワークに接続されたイベント駆動型分散制御マイクロコンピュータであり、前記コントロールクライアントで使用されるプログラム言語と異なる言語で記述されたプログラムを実行する複数の分散制御マイクロコンピュータと：前記第１の通信プロトコルにより前記第１通信ネットワークに接続されるとともに、前記第２通信プロトコルにより前記第２通信ネットワークを介して前記分散制御マイクロコンピュータを制御するローカルコントロールサーバであって、前記コントロールクライアントで使用される言語と同じ言語で記述されたプログラムを実行し、前記コントロールクライアントからダウンロードした処理プログラムが前記分散処理マイクロコンピュータで処理すべきプログラムの場合には、前記分散処理マイクロコンピュータで使用するプログラム言語に変換し、変換後の処理プログラムを前記分散処理マイクロコンピュータに配信するローカルコントロールサーバとで構成されることを特徴とする分散形制御ネットワークシステム。
【請求項２】　前記第１通信ネットワークに接続され、前記第１および第２通信ネットワークの各通信プロトコルを異なる通信プロトコルを介してコントロールサーバと複数の分散制御マイクロコンピュータとが接続された第３通信ネットワークを有し、前記第２通信ネットワークのコントロールサーバは前記第２通信ネットワークの通信プロトコルと第３通信ネットワークの通信プロトコルとを相互に変換する手段を有し、前記第２通信プロトコルに接続された分散制御マイクロコンピュータと、前記第３通信ネットワークに接続された分散制御マイクロコンピュータとを通信可能にしたことを特徴とする請求項１に記載の分散形制御ネットワークシステム。
【請求項３】　前記第１通信ネットワークは、ＴＣＰ／ＩＰの通信プロトコルにより定義されるグローバルネットワークであり、前記第２通信ネットワークはローカルネットワークであることを特徴とする請求項１に記載の分散形制御ネットワークシステム。
【請求項４】　第１の通信プロトコルにより第１通信ネットワークに接続されたコントロールクライアントと：第２の通信プロトコルにより第２通信ネットワークに接続された複数のイベント駆動型分散制御マイクロコンピュータであり、前記コントロールクライアントで使用されるプログラム言語と異なる言語で記述され、それぞれ割り当てられた異なるタスクを実行するコントロールノードと：前記複数の分散制御マイクロコンピュータのうち障害が発生した分散制御マイクロコンピュータが実行すべきタスクを別の分散制御マイクロコンピュータに実行させるように、障害が発生した分散制御マイクロコンピュータと、肩代わりさせるマイクロコンピュータとの対応関係が、前記コントロールクライアントで使用される言語と同じ言語で記述され、前記複数の分散制御マイクロコンピュータのいずれかに障害が発生すると、その障害が発生した分散制御マイクロコンピュータに割り当てられたタスクを別の分散制御マイクロコンピュータに自動的に割り当てることによりそのタスク処理を続行するように前記複数の分散制御マイクロコンピュータを制御するコントロールサーバとで構成されたことを特徴とする分散形制御ネットワークシステム。
【請求項５】　前記分散制御マイクロコンピュータの障害の発生は、前記コントロールクライアント側からは隠蔽されていることを特徴とする請求項４に記載の分散形制御ネットワークシステム。
【請求項６】　複数のコントロールノードと、これらのコントロールノードを制御するコントロールサーバとを有した分散制御ネットワークシステムにおいて、
被制御対象を表すアイコン群をグラフィックス表示画面上に表示するパーツウインドウと、
プログラム編集ウインドウと、
前記パーツウインドウに表示されたアイコンをドラグ・アンド・ドロップ操作により前記プログラム編集ウインドウに移動させ、前記被制御対象をどのノードに配置させるかを視覚的に組み合わせるとともに、前記複数のノード間を線で結ぶことにより処理の流れを定義する手段と、
前記定義された各ノードにおける被制御対象の配置とノード間での処理の流れに基づいて分散制御ネットワークシステムのためのソースプログラムを生成する手段とを具備することを特徴とするプログラム開発装置。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】この発明は、例えばJava等のインターネット用オブジェクト指向言語を用いた分散形制御ネットワークシステムに関する。
【０００２】
【従来の技術】分散形制御ネットワークシステムでは、図１０に示すように、インターネットプロトコルであるＴＣＰ／ＩＰを介して接続されるコントロールクライアント３やグローバルコントロールサーバ５のネットワークに加えて、ローカルコントロールサーバ７、１３を介して制御用の独自のネットワーク（ローカルコントロールネットワーク９、１５）が接続される。このローカルコントロールネットワーク９、１５には、独自の標準化されていない分散制御チップを搭載した複数のコントロールノード１１、１７が接続される。
【０００３】従来、このような分散形制御ネットワークシステムでは、同じローカルコントロールネットワーク内のコントロールノード間の制御のやりとりは可能だが、異なるローカルコントロールネットワークに接続されたコントロールノードとの制御のやり取りはできなか

った。すなわち、ローカルコントロールネットワーク９に接続されたコントロールノード１１からコントロールネットワーク１５に接続されたコントロールノード１７を制御することはできなかった（コントロールノード１７の実行環境に合わせたプログラミングを個別にする必要があった）。すなわち、個々のコントロールノードの処理内容を記述して、それぞれコンパイルしてオブジェクトコードを作成し、それらをリンカにより結合し、実行ファイルを作成していた。

【０００４】また、従来分散オブジェクト通信機構として、ＲＰＣ（Remote Procedure Call ）が知られている。ＲＰＣは分散処理を実現するための基本技術の１つであり、ネットワークに接続した別のコンピュータで動作する手続き（プログラム）を呼び出す。また、インターネット用オブジェクト指向言語「Ｊａｖａ」には分散オブジェクト通信機構としてＲＭＩ（Remote Method Invocation）が設けられている。しかし、これらの機構は、いずれも同期呼び出しであり、クライアントがサーバを呼び出し、サーバから結果が返ってくるまで、クライアント側は待ちの状態にあり、サーバから結果が返ってくるとクライアントが動き出す仕組みになっている。しかし、これは制御系のネットワークにはなじまない。なぜなら、制御系では、あるイベントの発生に応答してプログラムが動作し、何らかのイベントを発生させ、さらにそのイベントの発生に応答して別のプログラムが動作する性格のものだからである。

【０００５】一方、ＵＮＩＸの標準的なプログラム間通信機構として「ソケット」が知られている。しかしながら、「ソケット」を使う場合は、最初に、処理を依頼する側のプログラムと依頼される側のプログラムでやり取りするデータのフォーマットを、厳密に決めておく必要がある。これに合わせて双方のプログラムを記述する。相手のプログラムの処理手順を十分に理解していないと、そのプログラムに処理を依頼するプログラムは作れないという問題がある。また、「ソケット」は基本的に１対１のプログラム間通信機構である。

【０００６】

【発明が解決しようとする課題】上述したように従来は、分散形制御ネットワークシステムにおいて、異なるコントロールネットワークに接続されるコントロールノード間での通信ができなかった。

【０００７】また、プログラム間通信機構として、従来より知られているＲＰＣ，ＲＭＩは同期呼び出しのため、制御系にはなじまない。さらに、従来より知られている、「ソケット」は、データフォーマットが標準化されておらず、基本的に１対１のプログラム間通信のため、制御系になじまないという問題があった。

【０００８】また、オブジェクト指向用言語のプログラム開発ツールが種々開発されているが、ＧＵＩ（Graphical User Interface）環境を備えておらずプログラムを作るのが困難であった。例えば、各種ツールを使うときに、個々のツールの名前をいちいち入力しなければならず、使い勝手が悪かった。このため、容易にプログラムを作ることのできるプログラム開発ツールが求められていた。

【０００９】この発明の目的は、異なるローカルコントロールネットワークに接続されたコントロールノードを統一的に制御し、異なるネットワークに接続されたコントロールノード間での通信を可能にした分散形制御ネットワークシステムを提供することである。

【００１０】この発明の他の目的は、データフォーマットを標準化し、かつ１対複数のプログラム間通信機構を可能にする分散形制御ネットワークシステムを提供することである。

【００１１】この発明のさらに他の目的は、プログラムを容易に作成できる分散形制御ネットワークシステムを提供することである。

【００１２】

【課題を解決するための手段】この発明は、第１の通信プロトコルにより第１通信ネットワークに接続されたコントロールクライアントと；第２の通信プロトコルにより第２通信ネットワークに接続されたイベント駆動型分散制御マイクロコンピュータであり、前記コントロールクライアントで使用されるプログラム言語と異なる言語で記述されたプログラムを実行する複数の分散制御マイクロコンピュータと；前記第１の通信プロトコルにより前記第１通信ネットワークに接続されるとともに、前記第２通信プロトコルにより前記第２通信ネットワークを介して前記分散制御マイクロコンピュータを制御するローカルコントロールサーバであって、前記コントロールクライアントで使用される言語と同じ言語で記述されたプログラムを実行し、前記コントロールクライアントからダウンロードした処理プログラムが前記分散処理マイクロコンピュータで処理すべきプログラムの場合には、前記分散処理マイクロコンピュータで使用するプログラム言語に変換し、変換後の処理プログラムを前記分散処理マイクロコンピュータに配信するローカルコントロールサーバとで構成されることを特徴とする。

【００１３】また、この発明によれば、ローカルコントロールサーバで使用される言語はＪａｖａであり、前記ローカルコントロールサーバは、前記ダウンロードした処理プログラムが前記分散制御マイクロコンピュータで実行すべき処理プログラムの場合には、前記分散制御マイクロコンピュータの実行環境に合わせてジャスト・イン・タイム・コンパイルする手段を有することを特徴とする。

【００１４】また、この発明によれば、第１通信ネットワークに接続され、前記第１および第２通信ネットワークの各通信プロトコルを異なる通信プロトコルを介してコントロールサーバと複数の分散制御マイクロコンピュ

ータとが接続された第3通信ネットワークを有し、前記第2通信ネットワークのコントロールサーバは前記第2通信ネットワークの通信プロトコルと第3通信ネットワークの通信プロトコルとを相互に変換する手段を有し、前記第2通信プロトコルに接続された分散制御マイクロコンピュータと、前記第3通信ネットワークに接続された分散制御マイクロコンピュータとを通信可能にしたことを特徴とする。

【0015】また、この発明によれば、前記第1通信ネットワークは、TCP／IPの通信プロトコルにより定義されるグローバルネットワークであり、前記第2通信ネットワークはローカルネットワークであることを特徴とする。

【0016】また、この発明によれば、第1の通信プロトコルにより第1通信ネットワークに接続されたコントロールクライアントと；第2の通信プロトコルにより第2通信ネットワークに接続された複数のイベント駆動型分散制御マイクロコンピュータであり、前記コントロールクライアントで使用されるプログラム言語と異なる言語で記述され、それぞれ割り当てられた異なるタスクを実行するコントロールノードと；前記複数の分散制御マイクロコンピュータのうち障害が発生した分散制御マイクロコンピュータが実行すべきタスクを別の分散制御マイクロコンピュータに実行させるように、障害が発生した分散制御マイクロコンピュータと、肩代わりさせるマイクロコンピュータとの対応関係が、前記コントロールクライアントで使用される言語と同じ言語で記述され、前記複数の分散制御マイクロコンピュータのいずれかに障害が発生すると、その障害が発生した分散制御マイクロコンピュータに割り当てられたタスクを別の分散制御マイクロコンピュータに自動的に割り当てることによりそのタスク処理を続行するように前記複数の分散制御マイクロコンピュータを制御するコントロールサーバとで構成されることを特徴とする。

【0017】また、この発明によれば、分散制御マイクロコンピュータの障害の発生は、前記コントロールクライアント側からは隠蔽されていることを特徴とする。

【0018】また、この発明による分散形制御ネットワークシステムの開発ツールによれば、複数のコントロールノードと、これらのコントロールノードを制御するコントロールサーバとを有した分散形制御ネットワークシステムにおいて、被制御対象を表すアイコン群をグラフィックス表示画面上に表示するパーツウインドウと、プログラム編集ウインドウと、前記パーツウインドウに表示されたアイコンをドラグ・アンド・ドロップ操作により前記プログラム編集ウインドウに移動させ、前記被制御対象をどのノードに配置させるかを視覚的に組み合わせるとともに、前記複数のノード間を線で結ぶことにより処理の流れを定義する手段と、前記定義された各ノードにおける被制御対象の配置とノード間での処理の流れに基づいて分散制御ネットワークシステムのためのソースプログラムを生成する手段とを具備することを特徴とする。

【0019】この発明によれば、インターネットとローカルコントロールネットワークとを接続し、ルータとしての機能を果たすローカルコントロールサーバに、インターネットプロトコルであるTCP／IPと、ローカルコントロールネットワークの専用プロトコルとの変換機能を持たせている。さらに、ローカルコントロールサーバの制御内容をインターネット用オブジェクト指向言語である「Java」で記述できるように構成し、「Java」の実行環境を有していないコントロールノードに対しては、ローカルコントロールサーバが、そのコントロールノード用のコンパイラを有し、そのコントロールノードの実行環境に合わせてジャスト・イン・タイム・コンパイルし、ノードコントローラに配信する。

【0020】また、プログラム間通信において、ネットワーク変数を導入することによりデータフォーマットを標準化するとともに、ブロードキャストあるいはマルチキャストの通信が可能である。これにより、制御系がインターネットの世界に自然な形で取り込むことができる。

【0021】また、アプリケーションプログラムの作成は、プログラム部品を表すボタンをシステム設計用カンバス上にドラッグアンドドロップするだけでソースコードが生成可能であるため、ビジュアルなGUI環境下で容易にプログラム開発を行うことができる。

【0022】

【発明の実施の形態】以下、本発明の実施の形態につき図面を参照して説明する。

【0023】「第1の実施形態」図1は本発明の第1の実施形態に係る分散形制御ネットワークシステムのブロック構成図である。

【0024】図2において、複数のコントロールクライアント21a、21b、21cがインターネットのようなTCP／IPプロトコルを介して接続されている。さらに、コントロールクライアント21a、21b、21cからの処理要求を実行するグローバルコントロールサーバ25がTCP／IPを介してこれらのコントロールクライアント21a、21b、21cと接続される。さらに、グローバルコントロールサーバ25は、TCP／IPを介して複数のローカルコントロールサーバ27a、27bと接続される。ローカルコントロールサーバ27aはローカルコントロールネットワーク29を介して複数のコントロールノード31a、31b、31cと接続される。同様にしてローカルコントロールサーバ27bはローカルコントロールネットワーク33を介して複数のコントロールノード35a、35b、35c、35dと接続される。

【0025】コントロールノード31aは、分散制御マ

イクロコンピュータであり、このような分散制御マイクロコンピュータとしては、例えば米国Ｅｃｈｅｌｏｎ社製ＮＥＵＲＯＮチップ（東芝社製ニューロンチップＴＭＰＮ３１５０）が適用できる。この分散制御マイクロコンピュータのタスクのスケジューリングは、イベント駆動になっている。すなわち、ある特定の条件がＴＲＵＥになったとき、その条件にリンクしているコード（タスク）が実行される。例えば入力ピンの状態の変化、ネットワーク変数の新しい値の受信、タイマーの時間切れといった一定のイベントの結果として、特定のタスクが起動されるように定義される。コントロールノード間の通信はローカルな通信プロトコル、例えば米国Ｅｃｈｅｌｏｎ社のＬＯＮＴＡＬＫにより行われる。このＬＯＮＴＡＬＫプロトコルの詳細については、例えば、東芝ニューロンチップＴＭＰＮ３１５０／３１２０データブック（１９９５年９月）に記載されている。

【００２６】このコントロールノードに行わせるべき制御内容は、このコントロールノード専用言語、例えばＥｓｈｅｌｏｎ社のＮｅｕｒｏｎＣ言語で記述される。

【００２７】ローカルコントロールサーバ２７ａはこれらのコントロールノード３１ａ、３１ｂ、３１ｃを制御するためのローカルネットワークシステム（ＬＮＳ）を有する。なお、ローカルノード３１ａ，３１ｂ，３１ｃとローカルコントロールサーバ２７ａとで構成されるネットワークをローカルオペレーティングネットワーク（ＬＯＮ）と呼ぶことにする。

【００２８】図２はローカルコントロールサーバ２７ａのアーキテクチャを示す概念図である。図２に示すように、この発明の分散形制御ネットワークシステムにおいては、ローカルコントロールサーバ２７ａはＪａｖａＯＳ(Operating System)により制御される。アプリケーション４３は、個々のコントロールノード３１ａ、３１ｂ、３１ｃに実行させる機能をユーザがインターネット用オブジェクト指向言語Ｊａｖａで記述したものである。「Ｊａｖａ」はソフトとハードを明確に分け、マイクロプロセッサやＯＳを変えても再コンパイルせずに動作するプログラムを作成できるとともにバグの少ないプログラムを開発しやすいという利点を有する。このため、生産性や安全性（プログラムに内在するバグの可能性を低くする）を確保するとともにプロセッサアーキテクチャからの独立を実現したソフトウエア開発が可能となる特徴を有する。タスクマネージャである「ＬｏｎＴａｓｋＭａｎａｇｅｒ」３７は、個々のノードの実行環境に適合したコンパイラを有しており、上記ユーザがＪａｖａにより記述したアプリケーションプログラムをジャスト・イン・タイム・コンパイルして、個々のノードの実行環境に適合したオブジェクトコードを生成し、ＬＯＮの通信プロトコルである「Ｌｏｎ　Ｔａｌｋ」３９、およびＬＯＮの入出力制御を司る「Ｌｏｎ　ＩＯ」４１を用いて、コントロールノードに配信する。ジャスト・イン・タイム・コンパイラ方式は、グローバルコントロールサーバ２５からダウンロードしたバイト・コードを実行環境のＯＳとマイクロプロセッサ向けのコードに変換してから実行する方式である。すなわち、この実施例で言えば、ローカルコントロールサーバ２７ａは、ダウンロードしたバイト・コードをコントロールノードの実行言語であるＮｅｕｒｏｎＣ言語に変換する機能を有する。このような、構成にすることにより、ユーザは容易にプログラミングをすることが可能となる。すなわち、従来は、コントロールノードに実行すべき機能を例えばＮｅｕｒｏｎｎＣ言語を用いて記述し、さらにローカルコントロールサーバで実行すべき機能は例えばオブジェクト指向言語Ｃ＋＋で記述していた。そして、個々のノードのタスクを記述してオブジェクトコードとしてデータベースに格納しておき、必要に応じてタスクマネージャがデータベースから読み出してきて、対応するノードに配信し実行していた。このため、例えば、ノードを構成する制御チップが別のチップに置き換わった場合、すべてのプログラムを作り直さなければならない。一方、この発明によれば、ユーザは、コントロールノードを意識する必要は無く、Ｊａｖａ言語を用いてローカルコントロールサーバに対してのみプログラミングすればよいことになる。たとえ、ノードを構成する制御チップが別のチップに置き換わったとしても、ノードのプログラムを書き換える必要がない。

【００２９】さらに、この発明の実施例におけるコントロールノードには、複数のノード間でのデータの共有を簡単にするために、ネットワーク変数（ＮＶ）というオブジェクトを有する。ネットワーク変数とは別のノード上のネットワーク変数に接続されているオブジェクトである。ネットワークから見ると、ネットワーク変数はノードの入出力の定義になっており、分散アプリケーションでのデータ共有を可能にしている。あるプログラムがoutput（出力）ネットワーク変数に何か書き込めば、ネットワークを経由して、その出力ネットワーク変数に接続しているinput （入力）ネットワーク変数を持つすべてのノードにその値が伝達される。ネットワーク変数の伝達にはＬＯＮＴＡＬＫメッセージを使用し、メッセージの送信は自動的に行われる。すなわち、ネットワーク変数の値を更新するための送受信については、アプリケーションプログラムの明示的な操作は必要ない。ネットワーク変数を使用すると、個々のノードが独立に定義でき、新しいＬＯＮＴＡＬＫアプリケーションにノードを接続したり、再接続するのも簡単になるという機能である。しかし、従来は、出力ネットワークの変数の値は同一ネットワーク内のノードに対してブロードキャストされるが、異なるネットワークに接続されたノードにブロードキャストすることはできなかった。この発明によれば、ローカルコントロールサーバに、異なるネットワーク間のプロトコル変換機能を持たせることにより、ある

ネットワークに接続されたノードから異なるネットワークに接続されたノードへのブロードキャストが可能となる。

【0030】したがって、この発明の第2の特徴として、ローカルコントロールサーバ上で動く部分と、分散ノード間でやりとりする部分とを完全に切り離す形で決めることにより、ローカルコントロールサーバ上で動かすプログラム部分と、どのノードにどのプログラム部分を実行させるかという個々のプログラム部分の配置関係だけのコードを生成するだけで、所望の分散形システムネットワークを構築することができる。

【0031】上記構成によれば、例えばコントロールノード31aが故障した場合、コントロールノード31aで実行すべきタスクをコントロールノード31bで実行するようにあらかじめ、ローカルコントロールサーバ27aに定義しておくことにより、自動的に故障したコントロールノードのタスクが別の正常なコントロールノードに配信されて実行可能である。

【0032】図3は、ローカルコントロールサーバにNeuronChip用のジャスト・イン・タイム・コンパイラを持たせた実施形態を示す概念図である。同図に示すように、LonTaskのローディング後、LonTaskのインスタンスバイトコード（クラスのバイトコードではない）をNeuronジャストインタイムコンパイラを用いてジャスト・イン・タイム・コンパイルし、NeuronChip用のオブジェクトを生成する。この場合、LonTaskが実行時間内にダイナミックアロケーションやダイナミックローディングを必要としなければ、特別のランタイムを必要とすることなくNeuronChip用のオブジェクトコードを生成することができる。

【0033】図4はこの発明のプロセスコントロールネットワーク（PNC）の実行形態を示す概念図である。図4（a）に示す実行形態は、TCP／IPの通信プロトコルを介して接続されたPNCノードがクライアントからのプロシジャコール（procedure call）をタスクマネージャが実行する。図4（a）に示すPNCノードは例えばJavaで記述される。図4（b）に示す実行形態は、マネージメントノードとPNCノードとからなり、マネージメントノードはTCP／IPを介してクライアントと接続され、さらにPNCノードとは、非TCP／IP（例えばLonTalk）を介して接続されている。マネージメントノードおよびPNCノードは共にJavaで記述される。マネージメントノードは、クライアントからのプロシジャコールを受け取り、タスクマネージャスタブにより、PNCノードにタスクマネージャエージェント送り、実際のタスクの実行をPNCノードに行わせるが、クライアント側から見ると、あくまでもマネージメントノードが実行しているように見える。図4（c）に示す実行形態は、マネージメントノードがTCP／IPを介してクライアントと接続され、さらに非TCP／IP（例えばLonTalk）を介して、専用の分散制御チップ（例えばNeuronChip）と接続されている。マネージメントモードは例えばJavaで記述され、ニューロンチップは、専用言語（Neuron C）で記述されている。マネージメントノードはニューロンチップタスクマネージャを有し、クライアントからダウンロードしたタスクをニューロンチップ用のオブジェクトコードにジャスト・イン・タイム・コンパイルし、ニューロンチップにダウンロードする。ニューロンチップは、ニューロンチップファームウエアによりダウンロードされたオブジェクトコードを実行する。この場合にも、クライアント側にはあくまでもマネージメントノードが実行しているように見える。

【0034】上述したように、この発明の第1の実施形態によれば、LONの世界をJavaの世界にマッピングする。言い換えれば、LONのプログラミングモデルをJavaに拡張している。すなわち、Neuron Cで記述されたタスクは、Javaのタスククラスとして定義し、neuron Chip のスケジューラはJavaのタスクマネージャとして定義し、さらにネットワーク変数は、JavaのNVクラスとして定義する。この場合のプログラミングモデルの一例を以下に示す。

【0035】

```
LON World                          Programming Model
when(&lt;EVENT&gt;){          →    Java World
 &lt;TASK&gt;                          class Xtask extends LonTask {
}                                     public boolean when () {
                                     &lt;EVENT&gt;
                                    }
                                    public void task() {
                                       &lt;TASK&gt;
                                    }
                                  }
Network input int X;        →    NVRefInt X =
                                       new NVRefInt ("input");
10_ 7 input bit Y;          →    IORef Y =
```

|第2の実施形態| 図5はGUI (Graphical User Interface) 下でビジュアルにプログラム開発を行うためのツール (Visual Java Lon Builder) を概念的に示す図であり、複数のノード間にまたがる分散形制御ネットワークシステムの構築を簡単に開発することができる。

【0036】今、仮に図5において、スイッチがオンならば、ランプをオンにし、スイッチがオフならばランプをオフにするという制御を行いたいとする。この場合、ランプに対する制御をノード1で行い、スイッチに対する制御をノード2で行うとすると、各ノードに対する制御の記述は図6 (a) のようになる。図6 (a) はNeuron Cの言語で制御内容を記述したものである。この図6 (a) に示す制御内容をJavaで記述したものが図6 (b) である。すなわち、ユーザはスイッチの制御およびランプの制御についてあらかじめJavaで記述しておく。もちろん、これらを予め標準化タスクとして用意しておけば、ユーザは図6 (b) に示す記述を行う必要はない。そして、実行時にユーザはグラフィック画面上でアイコンの形で表されたパーツをドラッグアンドドロップ操作により、どのノードに配置するかを決定してやることにより、図6 (c) に示すようなその配置を表すメインプログラムの自動生成が可能である。

【0037】以下、プログラム例について図7乃至図9を参照して説明する。今、図7に示すように3つのノードがあり、各ノードに1ずつタスクが配置されているものとする。第1ノードはIOピン2から、スイッチの状態をよみだし、ネットワーク変数" Switch" に結果を出力する。第2ノードはIOピン4から読んだ状態と、ネットワーク変数" Switch" の排他論理和をとり結果を、ネットワーク変数" Switch2" に出力する。第3ノードはネットワーク変数" Switch2" の状態をIOピン3に出力する。

【0038】以上がプログラマに直接見えるプログラム

```
Public class Lon{
  public static void main(String [] args){
// network variables global definition
//ネットワーク変数オブジェクトの生成
   NetVarInt nv _switch1 = new NetVarInt("Switch1");
   NetVarInt nv _switch2 = new NetVarInt("Switch2");
// Lon node 1 -- remote switch1 module
  LonTaskManager node1 = new LonTaskManager("Node1");
//タスクマネージャオブジェクトを生成する。
```

【0048】//" Node1" という名前のノードオブジェクトを検索し

```
//これと、生成したオブジェクトを関連づける。
```

【0049】

```
   NetVarRefInt nv sw1_out = new NetVarRefInt(node1, "output",nv_switch1
);
//ネットワーク変数" Switch1" に対して、値を出力するための
//ネットワーク変数参照オブジェクトを生成し、" Node1" と関連づける。
```

【0050】

```
new IORef("input", 7,"bit");
```

(ユーザがプログラムを行う部分) である。これ以外の部分は、クラス定義の中に隠蔽されており、プログラマは意識する必要がない。

【0039】import java.awt.*;
これはメインプログラムであり、ノード上でタスクの制御を行う以下の処理を行う。

【0040】1. LonTaskManagerクラスのオブジェクト生成。ノード名からノードオブジェクトを検索し、関連付けを行う。

【0041】2. ノード間のデータのやりとりを仲介するネットワーク変数 (NetVarInt) クラスのオブジェクト生成。

【0042】3. ノード上でタスクとネットワーク変数オブジェクトとのインターフェースを行う。ネットワーク変数参照 (NetVarRefInt) クラスのオブジェクト生成とネットワーク変数および、ノードとの関連づけ。

【0043】4. ノード上のタスクと、IOオブジェクトとのインターフェースを行うIO参照 (IORef) クラスのオブジェクト生成を行い、ノードとの関連づけを行う。(IOオブジェクト自体は、ノードオブジェクトに隠蔽されており、IOピン番号を指定することによって、対応づけられる)

5. タスクを生成し、ノード、ネットワーク変数参照、IO参照の各オブジェクトと関連付ける。

【0044】以上で、プログラムを実行する準備が完了する。

【0045】6. 各ノードに対して、タスク実行の指令をだす。

【0046】このプログラムの修正のみで、タスク内部のプログラムは一切変えず、ノード、ネットワーク変数、IO、および、これらとタスクとの関係を変更することができる。

【0047】

```
  IORef io sw1 = new IORef(node1,"input",2);
//ＩＯピン2からデータを入力するためのＩＯ参照オブジェクトを生成し
//"Ｎｏｄｅ1"と関連づける。
```

【0051】

```
 TaskSwitch switch1 _task = new TaskSwitch("Switch1",node,nv_sw1_out,
io_sw1);
//TaskSwitchクラスのタスクを生成し、ノード、ネットワーク変数、ＩＯとの関
連づけを行う。
```

【0052】

```
//Lon node 2 -- remote switch2 module output (switch1 EXOR switch2)
  LonTaskManager node2 = new LonTaskManager("Node2");
  NetVarRefInt nv _sw1_in = new NetVarRefInt(node2,"input",nv _switch1);
  NetVarRefInt nv _sw2_out = new NetVarRefInt(node2, "output", nv _switc
h2);
  IORef io_sw2 = new IORef(node2, "input",4);
  TaskExor switch2_task = new TaskExor("Switch2",node2,
  nv_sw1_in, nv _sw2_out,io _sw2);
//Lon node3 -- remote lamp module
  LonTaskmanager node3 = new LonTaskManager("Node3");
  NetVarRefInt nv _sw2_in  = new NetVarRefInt(node3, "input",nv _switch2
);
  IORef io_lamp = new IORef(node3, "output",3);
  TaskLamp lamp _task = new TaskLamp("Lamp", node3,nv _sw2_in,io_lamp);
// Task start
   node1. start();
   node2. start();
   node3. start();
//各々のノードに関連づけられて、タスクの実行を開始する。
```

【0053】//ノードの属性のうち、実行される計算機を指定する属性がメインプログラムが
//実行されている計算機と異なる場合、属性で指定された計算機上に
//各オブジェクト（タスクマネージャ、タスク、ネットワーク変数参照、
//ＩＯ参照）が転送され、ＲＭＩによって、起動がかけられる。

【0054】//　また、ノードの属性が、タスクの実行のためにコンパイラを必要とする
//場合、対応するＪＩＴ（Just-In-Time compiler ）が起動され、ネイティブオブジェクトに変換された後、ダウンロードされ、実行が開始される。

【0055】

```
    }
  }
  /*タスク1
```

ＩＯオブジェクトから、データ（スイッチの状態、ST_ON またはST_OFF）を読み出し、値をネットワーク変数に書き出す。対象となる、ＩＯオブジェクトおよび、ネットワーク変数オブジェクトは、タスク生成時（インスタンス生成時）、パラメータで渡される。

【0056】

```
*/
Class TaskSwitch extends LonTask {
  private NetVarRefInt SwitchOut;
  private IORef io switch;
      //ネットワーク変数参照オブジェクト、ＩＯ参照オブジェクトを
      //アクセスするための、オブジェクトローカル変数スロット
public boolean when () {
  return (io_changes(io _switch) );
}
//whenメソッド。タスクマネージャは、定期的にこのメソッドを呼び出し値が真
```

```
            //のとき、下記のtaskメソッドの内容を実行する。
【0057】//IO_changesは，ＩＯオブジェクトの状態      【0058】
が変化したとき、真を返す。
            //public void task () {
              if (io in(io_switch) ==ST _ON)
               SwitchOUt.set(ST _ON);
             else
               SwitchOut.set (ST_OFF);
            }
            //taskメソッド。whenメソッドが真を返したとき、ここが実行される。
【0059】//io _in は、ＩＯオブジェクトの状態値        によって、値が書き込まれる。
を読み出す。                                          【0061】
【0060】// ネットワーク変数参照のset メソッド
            TaskSwitch(String myname, LonTaskManager node, NetVarRefInt nv_sw, IORef
             io _sw) {
              super(myname, node);
              SwitchOut = nv_sw;
              io_switch = io_sw;
            }
                 // タスクオブジェクトのコンストラクタ。インスタンス生成時のパラメー
            タ
                 // として対応づけられるノードおよびネットワーク変数、ＩＯがきまる。
                                                  【0062】
                 // ノード（タスクマネージャ）との関連づけは、上位クラス(LonTask) の
                 // コンストラクタで行われる。（super(myname, node) のところ。）
            }
            /*タスク２
ＩＯオブジェクトから、データ（スイッチの状態、ST_O      スク生成時（インスタンス生成時）、パラメータで渡さ
N またはST_OFF）を読み出し、ネットワーク変数オブジ      れる。
ェクトの状態と排他論理和をとる。対象となる、ＩＯオ      【0063】
ブジェクトおよびネットワーク変数オブジェクトは、タ
            */
            class TaskExor extends LonTask{
              private NetVarRefInt SwitchIn;
              private NetVarRefInt SwitchOut;
              private IORef io_switch;
              public boolean when () {
                return (io_changes(io _switch)  || nv _update _occur(SwitchIn) );
            }
                 //nv _update _occurは、ネットワーク変数オブジェクトの
                 // 値が更新されたとき、真を返す。
【0064】
            public void task () {
              if ( (IO_in(io_switch)^SwitchIn.state()) == ST_ON)
                SwitchOut.set(ST_ON);
              else
                SwitchOut.set(ST_OFF);
            }
            //ネットワーク変数参照オブジェクトのstate メソッドは、
            //ネットワーク変数値を返す。
```

【0065】

```
TaskExor(String myname, LonTaskManager node, NetVarRefInt nv _sw _in, N
etVarRefInt nv_sw _out, IORef io_sw) {
   super(mayname, node);
   SwitchIn = nv_sw _in;
   SwitchOut = nv _sw _out;
   io _switch = io_sw;
 }
}
/*タスク3
```

ネットワーク変数オブジェクトの状態をＩＯオブジェクトに出力する。

【0066】

```
 */
 class TaskLamp extends Lontask {
  private NetVarRefInt SwitchIn;
  private IORef LampOut;
  public boolean when () {
     return (nv _update _occur(SwitchIn) );
 }
 public void task () {
  io_out(LampOut, SwitchIn, state());
 }
  //io_outは、ＩＯオブジェクトに値を書き込む。
```

【0067】

```
TaskLamp( String myname, LonTaskManager node, netVarRefInt nv _sw, IORef
 io _lamp) {
  super(myname, node);
  SwitchIn = nv _sw;
  LampOut = io_lamp;
}
```

【0068】

【発明の効果】以上述べたように、この発明によれば、（１）個々のアーキテクチャの違いをローカルコントロールサーバで吸収することにより異なるローカルネットワークに接続されたノード間での通信が可能になる。（２）また、個々のノード間の通信に使用されるネットワーク変数を、ローカルコントロールサーバにも用いることにより、プログラミングを容易にするとともにブロードキャストあるいはマルチキャストのプログラム間通信を実現可能である。（３）さらに、ビジュアルなＧＵＩ環境下でネットワークシステムの設計を可能とすることにより、プログラミングの手間を省き、デバグ効率を高め、容易にプログラム開発を行うことができる。（４）Ｊａｖａは動的リンクを採用し、リンク単位である.classファイルは基本的に１つのクラスとそのメソッドを格納するので、柔軟なシステムが構築可能である。また、ＪａｖａプログラムがＷ新しいプラットフォームで動作するにはＪａｖａＶＭだけを移植すればよいので、一度移植してしまえばどんなＪａｖａプログラムでも移植のために作り直す必要はない。また、仮想マシンを移植する労力は、伝統的なプログラミング言語での労力（個々のプログラムとライブラリの稼動環境に依存する部分を個別に変更して、さらに再コンパイルする作業）に比べて小さい。したがって、柔軟な分散形制御ネットワークシステム構成が可能である。さらに、Ｊａｖａはインターネットからダウンロードしたコードを実行する環境を想定しているので、プログラムの暴走を避けるためのエラーチェック機能を有するため、分散形制御ネットワークシステムの安全な実行が可能となる。

【図面の簡単な説明】

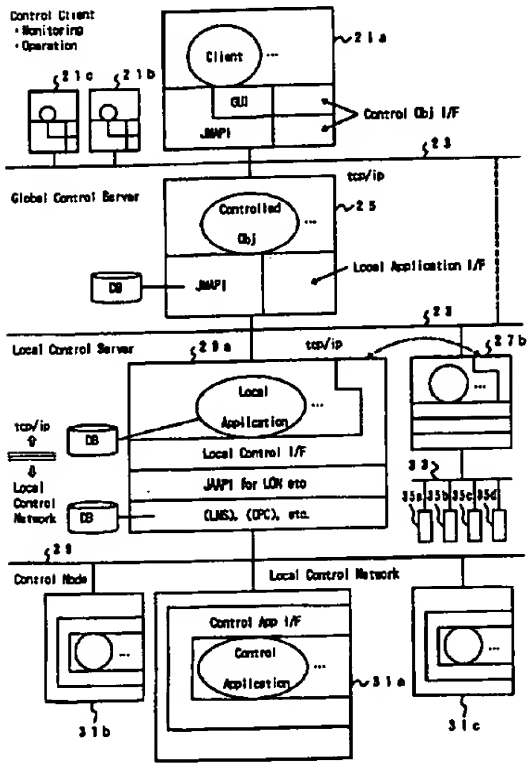
【図１】この発明の第１の実施形態に係る分散形制御ネットワークシステムのブロック図。

【図２】図１に示すローカルコントロールサーバ内のプロセスネットワークコンピュータアーキテクチャを示す概念図。

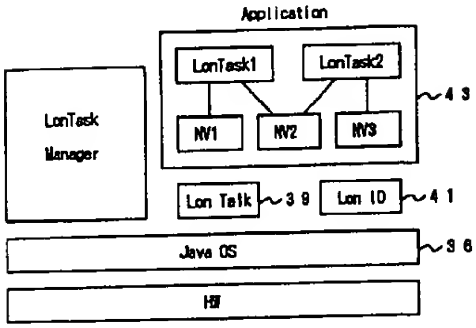
【図３】図１に示すローカルコントロールサーバによって行われる、個々のコントロールノードに割り当てられたプログラムのクラスに対するジャスト・イン・タイム

コンパイルを示す概念図。
【図4】この発明のプロセスコントロールネットワークの実行形態を示す概念図。
【図5】この発明の第2の実施形態であるビジュアルなGUI環境下でのシステム開発ツールの一例を示す概念図。
【図6】図5に示す各ノードに対する制御内容を記述したソースコードリスト。
【図7】この発明により分散形制御ネットワークシステムにおけるプログラミングにおいて、各ノードでのタスクの定義及びノード間での処理の流れを視覚的に示す概念図。
【図8】図7に示すタスクの定義およびノード間での処理の流れの実際のコーディング例を示すプログラムリストの一部。
【図9】図7に示すタスクの定義およびノード間での処理の流れの実際のコーディング例を示すプログラムリストの残りの部分。
【図10】一般的な分散形制御ネットワークシステムを示すブロック図。
【符号の説明】
1…ネットワーク
3…コントロールクライアント
5…グローバルコントロールサーバ
7…ローカルコントロールサーバ
9…ローカルコントロールネットワーク
11…コントロールノード
13…ローカルコントロールサーバ
15…ローカルコントロールネットワーク
17…コントロールノード
21…コントロールクライアント
23…TCP／IP
25…グローバルコントロールサーバ
27…ローカルコントロールサーバ
29…ローカルコントロールネットワーク
31…コントロールノード
33…ローカルネットワーク
35…ローカルノード
36…JavaOS
37…LonTaskManager
39…Lon Talk
41…Lon IO
43…アプリケーション

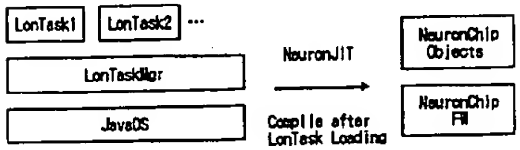
【図1】



【図2】

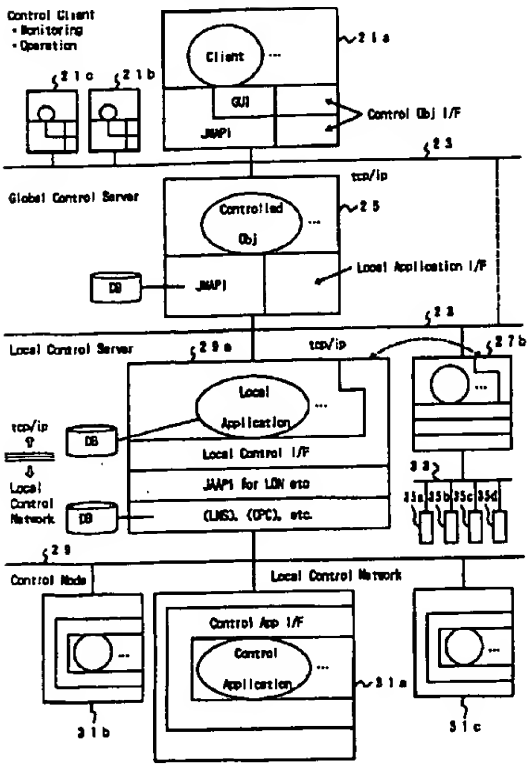


【図3】

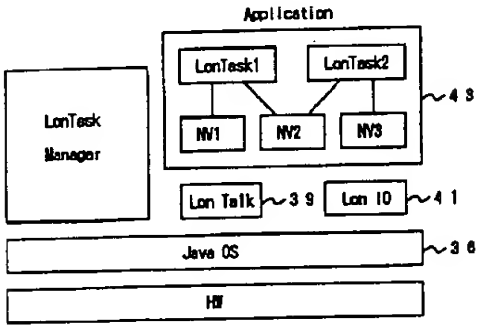


コンパイルを示す概念図。
【図4】この発明のプロセスコントロールネットワークの実行形態を示す概念図。
【図5】この発明の第2の実施形態であるビジュアルなGUI環境下でのシステム開発ツールの一例を示す概念図。
【図6】図5に示す各ノードに対する制御内容を記述したソースコードリスト。
【図7】この発明により分散形制御ネットワークシステムにおけるプログラミングにおいて、各ノードでのタスクの定義及びノード間での処理の流れを視覚的に示す概念図。
【図8】図7に示すタスクの定義およびノード間での処理の流れの実際のコーディング例を示すプログラムリストの一部。
【図9】図7に示すタスクの定義およびノード間での処理の流れの実際のコーディング例を示すプログラムリストの残りの部分。
【図10】一般的な分散形制御ネットワークシステムを示すブロック図。
【符号の説明】
1…ネットワーク
3…コントロールクライアント
5…グローバルコントロールサーバ
7…ローカルコントロールサーバ
9…ローカルコントロールネットワーク
11…コントロールノード
13…ローカルコントロールサーバ
15…ローカルコントロールネットワーク
17…コントロールノード
21…コントロールクライアント
23…TCP／IP
25…グローバルコントロールサーバ
27…ローカルコントロールサーバ
29…ローカルコントロールネットワーク
31…コントロールノード
33…ローカルネットワーク
35…ローカルノード
36…JavaOS
37…LonTaskManager
39…Lon　Talk
41…Lon　IO
43…アプリケーション

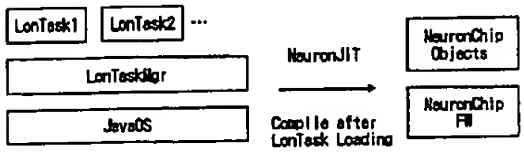
【図1】



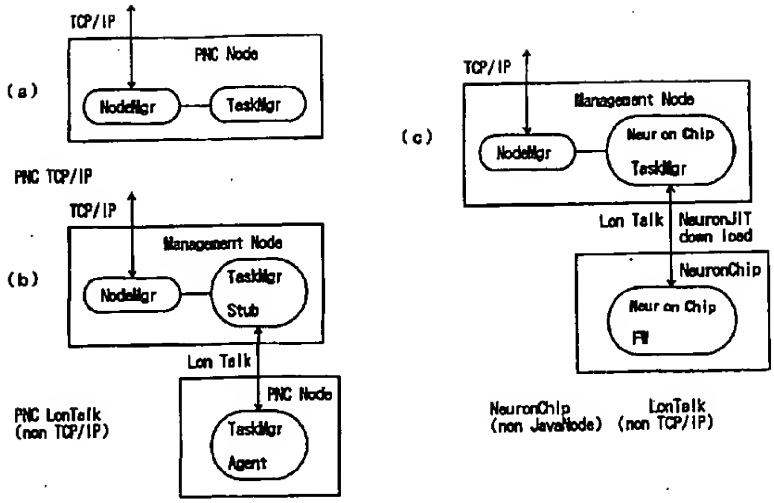
【図2】



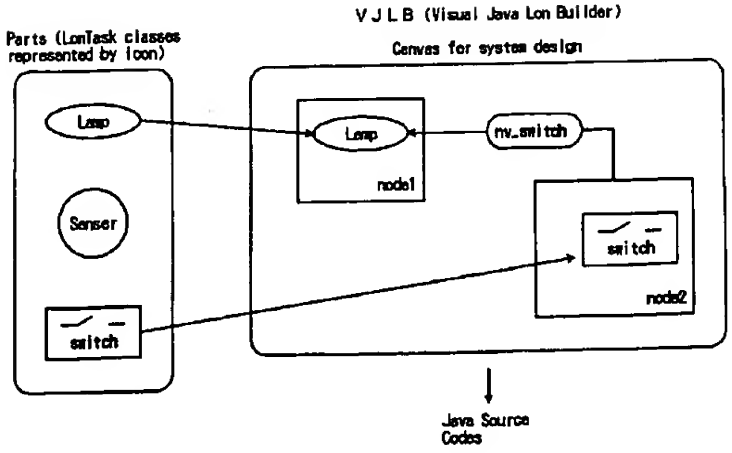
【図3】



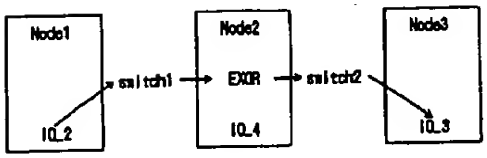
【図4】


(a)
TCP/IP
PNC Node
NodeMgr
TaskMgr
(b)
PNC TCP/IP
TCP/IP
Management Node
NodeMgr
TaskMgr
Stub
Lon Talk
PNC Node
TaskMgr
Agent
PNC LonTalk
(non TCP/IP)
(c)
TCP/IP
Management Node
NodeMgr
Neuron Chip
TaskMgr
Lon Talk
NeuronJIT
down load
NeuronChip
Neuron Chip
FW
NeuronChip
(non JavaNode)
LonTalk
(non TCP/IP)

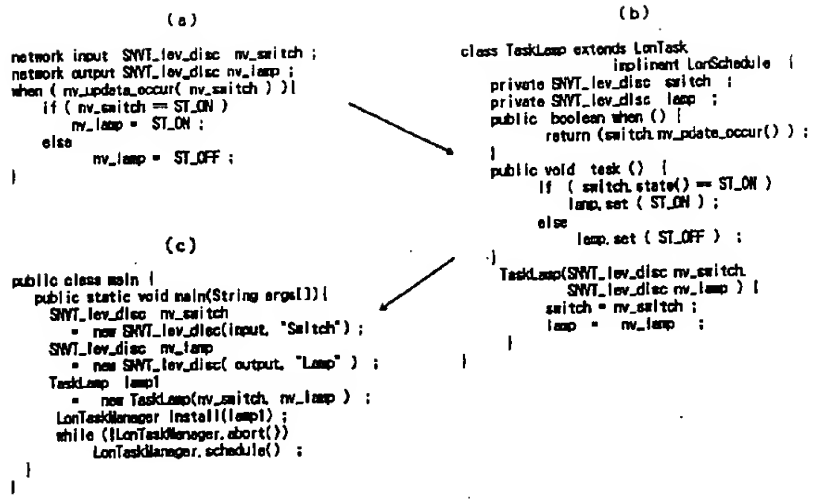
【図5】


VJLB (Visual Java Lon Builder)
Parts (LonTask classes represented by icon)
Canvas for system design
Lamp
Sensor
switch
Lamp
mv_switch
node1
switch
node2
Java Source Codes

【図7】


Node1
IO_2
switch1
Node2
EXOR
IO_4
switch2
Node3
IO_3

【図6】



(a)

```
network input  SNVT_lev_disc  nv_switch ;
network output SNVT_lev_disc nv_lamp ;
when ( nv_update_occur( nv_switch ) )
    if ( nv_switch == ST_ON )
        nv_lamp = ST_ON ;
    else
        nv_lamp = ST_OFF ;
}
```

(b)

```
class TaskLamp extends LonTask
                implinent LonSchedule {
    private SNVT_lev_disc  switch  ;
    private SNVT_lev_disc  lamp  ;
    public  boolean when () {
        return (switch.nv_pdate_occur() ) ;
    }
    public void  task ()  {
        if  ( switch.state() == ST_ON )
            lamp.set ( ST_ON ) ;
        else
            lamp.set ( ST_OFF )  ;
    }
     TaskLamp(SNVT_lev_disc nv_switch
            SNVT_lev_disc nv_lamp ) {
         switch = nv_switch ;
         lamp =   nv_lamp   ;
     }
}
```

(c)

```
public class main {
   public static void main(String args[]){
     SNVT_lev_disc  nv_switch
        = new SNVT_lev_disc(input, "Switch") ;
     SNVT_lev_disc  nv_lamp
        = new SNVT_lev_disc( output, "Lamp" ) ;
     TaskLamp  lamp1
        =   new TaskLamp(nv_switch, nv_lamp ) ;
      LonTaskManager install(lamp1) ;
      while (!LonTaskManager.abort())
          LonTaskManager.schedule()  ;
   }
}
```

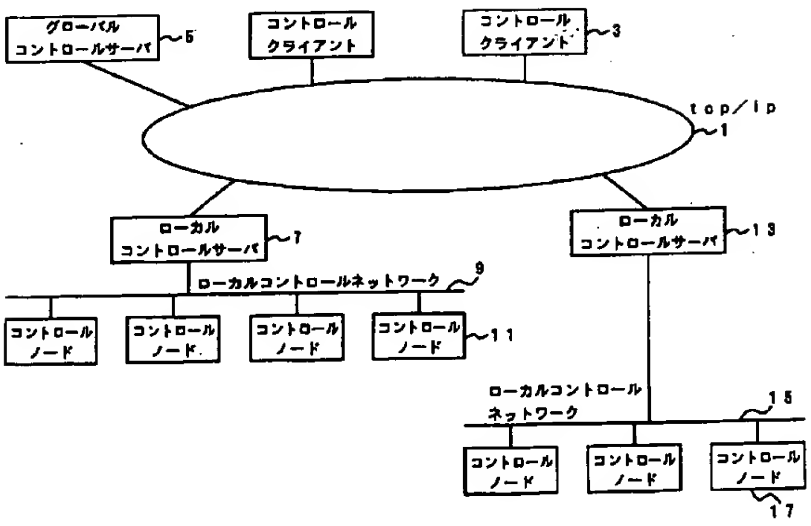
【図8】

```
import java.awt.*;
 public class Lon1
  public static void main( String [] args){

 // network variables global definition
   NetVarInt nv_switch1 = new NetVarInt("Switch1");
   NetVarInt nv_switch2 = new NetVarInt("Switch2");

// Lon node 1 -- remote switch1 module
  LonTaskManager node1 = new LonTaskManager("Node1");
  NetVarRefInt nv_sw1_out = new NetVarRefInt(node1, "output",nv_switch1);
  IORef io_sw1 = new IORef(node1, "input",2);
  TaskSwitch switch1_task = new TaskSwitch("Switch1",node1,nv_sw1_out,io_sw1);

// Lon node 2 -- remote switch2 module output (switch1 EXOR switch2)
  LonTaskManager node2 = new LonTaskManager("Node2");
  NetVarRefInt nv_sw1_in = new NetVarRefInt(node2,"input",nv_switch1);
  NetVarRefInt nv_sw2_out  = new NetVarRefInt(node2,"output",nv_switch2);
  IORef io_sw2 = new IORef(node2,"input",4);
  TaskExor switch2_task = new TaskExor("Switch2",node2,nv_sw1_in,nv_sw2_out,io_sw2)
;

// Lon node 3 -- remote lamp module
  LonTaskManager node3 = new LonTaskManager("Node3");
  NetVarRefInt nv_sw2_in = new NetVarRefInt(node3,"input",nv_switch2);
  IORef io_lamp = new IORef(node3,"output",3);
  TaskLamp lamp_task = new TaskLamp("Lamp",node3,nv_sw2_in,io_lamp);

// Task start
   node1.start();   node2.start();  node3.start();
 }
}
```

【図9】

```
class TaskSwitch extends LonTask {
 private NetVarRefInt SwitchOut;
 private IORef io_switch;

 public boolean when(){
  return ( io_changes(io_switch));
 }
 public void task(){
  if( io_in(io_switch) == ST_ON )
    SwitchOut.set( ST_ON );
  else
    SwitchOut.set( ST_OFF );
 }
 TaskSwitch(String myname,LonTaskManager node,NetVarRefInt nv_sw, IORef io_sw ){
   super(myname,node);
   SwitchOut = nv_sw ; io_switch = io_sw ;
 }
}

class TaskExor extends LonTask{
 private NetVarRefInt SwitchIn;
 private NetVarRefInt SwitchOut;private IORef io_switch;

 public boolean when(){
  return ( io_changes(io_switch)||nv_update_occur(SwitchIn));
 }
 public void task(){
  if( (io_in(io_switch)^SwitchIn.state()) == ST_ON )
    SwitchOut.set( ST_ON );
  else    SwitchOut.set( ST_OFF );
 }
 TaskExor(String myname,LonTaskManager node,
       NetVarRefInt nv_sw_in, NetVarRefInt nv_sw_out, IORef io_sw ){
  super(myname,node) ;
  SwitchIn = nv_sw_in ; SwitchOut = nv_sw_out ; io_switch = io_sw ;
 }
}

class TaskLamp extends LonTask {
 private NetVarRefInt SwitchIn;
 private IORef LampOut;

 public boolean when(){
  return ( nv_update_occur(SwitchIn) );
 }
 public void task(){
  io_out(LampOut,SwitchIn.state());
 }
 TaskLamp( String myname, LonTaskManager node,
           NetVarRefInt nv_sw, IORef io_lamp){
  super(myname,node);
  SwitchIn = nv_sw;  LampOut = io_lamp ;
 }
}
```

【図10】



422115

Differentiated Service for Telnet 3270 Services

When a Telnet 3270 (TN3270) Client exchanges files with an SNA Host acro ss a Telnet 3270 Server, the packets of information related to the file being exchanged are sent between the Client and its Server using the same sessions (TCP and SNA sessions) than those used for screen related interactions such as a screen refresh. This means tha t across the IP network, data packets related to the transfe r of files are using the same priority than the priority used for the interactive traffic due to screen related interactions. In a se nse, this mode of operation violates SNA fundamental rules associated with SNA Class Of Service (COS) and Transmission Priority (TP) within SNA networks. These rules assume that interactive traffic (such as screen related interactions) gets a better treatment within the network than lower priority traffic such as batch traffic (file transfer related traffic).

Transfer of large files between Clients and their Servers generate a huge amount of data packets that should not flow on the same pri ority as the interactive, screen related traffic. As a differentiation is not made, overall response time is severely impacted as now , the true interactive traffic but also the non-SNA data traffic will be penalized by a flooding of data packets related to the excha nge of files across the network. Exchanging files consumes bandwidth and processing resources within the network. In fact, from an SN A standpoint, the packets related to the exchange of files between a Client and its Server should have a lower priority than the real interactive traffic allowing the higher priority traffic to get a better service than any other low priority traffic.

Patent FR 9 99 90 21, "Method and System for improving overall network response time during the exchange of files between Telnet 3270 Servers and Telnet 3270 Clients", by Didier Giroir and Jean Lorrain desc ribes a set of mechanisms allowing TN3270 Clients and TN3270 Servers to dynamically specify a lower IP priority for the traffic relat ed to the exchange of files, than the priority associated to the traffic related with screen interactive traffic.

The Client and the associated Server specify an IP transport priority (Precedence / Type Of Service) for the data packets exchanged o ver TN3270 TCP sessions. The Client, when performing the transfer of a file to or from the SNA Host over the Telnet session dynamical ly changes (i.e downgrade) the IP priority associated to the interactive screen traffic for all data packets flowing to the Telnet 32 70 Server during the whole duration of the file download. The associated server, on receipt of a data packet from the client, memoriz es the associated priority and uses this priority over all packets that it sends back to the client. This allows network implementati on to give higher priority treatment to screen related data packets (interactive traffic) over data packets related to file transfer (batch traffic). The priority used by data packets related to file transfer is customisable, to give maximum freedom to the network d esigner. For example, SNA Screen traffic could flow on the highest priority, pure IP traffic (i.e not related to Telnet 3270) could f low on any of the medium priorities possible and data packet due to file transfer could use the lowest priority within the network.

This paper proposes to allow the Telnet 3270 Server to provide a differentiated treatment to data packets received from the Telnet 32 70 Clients, depending of their type, interactive (data packet related to screen session) or batch (data packets related to file trans fer between a Telnet 3270 Client and a Host). The same mechanism could be easily generalized to provide multiple priorities (i.e more than two).

The mechanism proposed here reuses the IP transport Priority as defined in Patent FR 9 99 90 21, the Telnet 3270 Server can exploit t his information to provide a differentiated treatment for the Telnet Session that currently perform Screen related interactions versu s those sessions that are involved with a transfer of file between the SNA Host and the Telnet 3270 Client. Alternatively, one may co nsider the use of a reserved field within the Telnet 3270 Header for the Server to make the difference between interactive and Batch session. In such an implementation, there is no need for the IP layer to signal to upper layer the related TOS of the received packet (which means the IP layer does not need to be modified). On receipt of a data packet, the Telnet 3270 processor can decide, based on the additional information within the TN3270 header, the relative priori ty of this data packet versus other data packets.

Whatever of the two means described above is chosen (the Telnet Server uses the IP packet transport priority or the added bit within the Telnet 3270 header), a mechanism based on multiple queues with a Service Order Table (with for example an aging mechanism) or any other algorithm can be implemented to provide differentiated treatment o f interactive versus batch traffic by the Telnet Server.

Disclosed by International Business Machines Corporation
422115